CSCE 230 – Lab 4: Assembly Subroutines

Due: 11:59PM, Sep 19 (Tuesday)

Objectives

- Learn how to use the stack and subroutines

Useful References on Canvas

- Lecture notes for Chapter 2 and Appendix B
- Altera Nios II Processor Document
- Altera Nios II Instruction Document

Lab: Assembly Subroutines

First, please create a new project as before. Note that, for the memory settings, **set .text to 0 and set .data to 0x400** (you do not need to type 0x) **for this lab**.

Then, write a **single** assembly program using **exactly the following template** to complete the following two tasks. **Do not make any changes to the main program and the data section**.

```
    .text
    .global  _start

# Main program
_start:
    movia   sp, 0x500

#  used to test whether your subroutines modify these registers
    movi    r2, 2
    movi    r3, 3
    movi    r4, 4
    movi    r5, 5
    movi    r6, 6
    movi    r7, 7
    movi    r8, 8
    movi    r9, 9
    movi    r10, 10
    movi    r11, 11
    movi    r12, 12
    movi    r13, 13
    movi    r14, 14
    movi    r15, 15
    movi    r16, 16
    movi    r17, 17
    movi    r18, 18
    movi    r19, 19
    movi    r20, 20
    movi    r21, 21
    movi    r22, 22
    movi    r23, 23
```

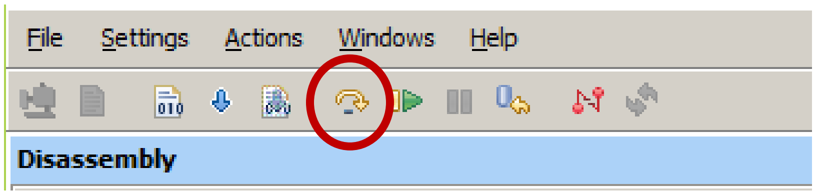
```
# used to test your function sum
    movia   r2, array1
    subi    sp, sp, 4
    stw     r2, (sp)             # push address of array1
    ldw     r2, array1_size(r0)
    subi    sp, sp, 4
    stw     r2, (sp)             # push size of array1
    subi    sp, sp, 4            # space for return value
    call    sum
    ldw     r2, (sp)             # get return value
    addi    sp, sp, 12           # pop input and return

# used to test your function compare
    movia   r3, array1
    subi    sp, sp, 4
    stw     r3, (sp)             # push address of array1
    ldw     r3, array1_size(r0)
    subi    sp, sp, 4
    stw     r3, (sp)             # push size of array1
    movia   r3, array2
    subi    sp, sp, 4
    stw     r3, (sp)             # push address of array2
    ldw     r3, array2_size(r0)
    subi    sp, sp, 4
    stw     r3, (sp)             # push size of array2
    subi    sp, sp, 4            # space for return value
    call    compare
    ldw     r3, (sp)             # get return value
    addi    sp, sp, 20           # pop input and return

end:br      end

# write your function sum below
sum:
    ....

    ret

# write your function compare below
compare:
    ....

    ret

# data section is located at address 0x400
    .data

array1:
    .word 1, 3, 5, 7, 9, 11, 13
array1_size:
    .word 7

array2:
    .word 16, 17, 18
array2_size:
    .word 3
```

Two Tasks

- Task 1: Please write a subroutine called *sum* to calculate the sum of all the elements of an array, which will be called in task 2.
    - Subroutine name: *sum*
    - Two input parameters (A, m): the address of array A, and the size m of array A (i.e., number of elements).
    - One return value: the sum of all elements of the array
    - All numbers (array element, input parameters, and return values) are 32-bit unsigned binary numbers.
    - Requirement: Your subroutine should pass all input parameters and return value using the stack
    - **Requirement: Your subroutine should not change any general-purpose registers such as r2, r3, r4, ..., sp, ra. That is, please save the original values of all registers using the stack at the beginning of your subroutine, and restore them at the end of your subroutine.**
    - Note: please read the main program to find the order of the input parameters and return value in the stack
- Task 2: Please write a subroutine called *compare* to compare the sums of two arrays
    - Subroutine name: *compare*
    - Four input parameters (A, m, B, n): the address of array A, and the size m of array A, the address of array B, and the size n of array B,
    - One return value:
        - return 0, if *sum*(A, m) <= *sum*(B, n)
        - return 1, otherwise
    - All numbers (array element, input parameters, and return values) are 32-bit unsigned binary numbers.
    - Requirement: This subroutine must call subroutine *sum* twice.
    - Requirement: This subroutine should pass all input parameters and return value using the stack.
    - **Requirement: Your subroutine should not change any general-purpose registers such as r2, r3, r4, ..., sp, ra. That is, please save the original values of all registers using the stack at the beginning of your subroutine, and restore them at the end of your subroutine.**
    - Note: please read the main program to find the order of the input parameters and return value in the stack

Quick correctness check:

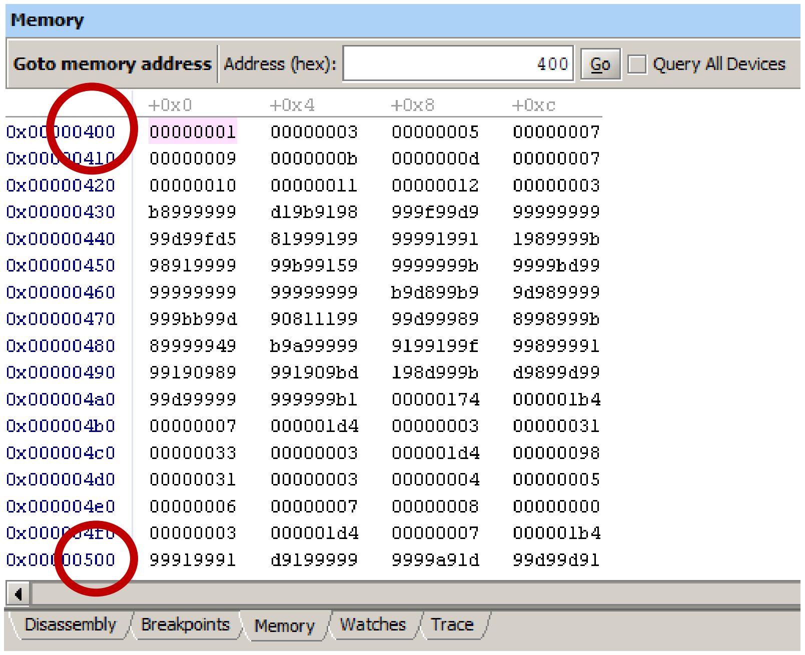
- Subroutines and stacks are hard to debug. The best way to find your mistakes is to run it step by step from the first instruction to the last one, and check corresponding register values and stack values.



- Right after you execute instruction “`ldw r2, (sp)`” (after “`call sum`”)
  - register r2 should be 0x31
  - register r3 should be 0x3
  - register sp should be 0x4f4
- After you execute instruction “br end”
  - register r2 should still be 0x31
  - register r3 should be 0
  - register r4 should be 4
  - register r5 should be 5
  - register r22 should be 22
  - register r23 should be 23
  - register sp should be 0x500

Finally, please answer the questions on Canvas, where you need to upload two screenshots **after running your program (i.e., set a break point at “br end”, and then run)**.

- Screenshot 1 of your data section and stack: Go to the “Memory” tab, type “400” into the “Address (hex)” box, and then click the “Go” button. Make sure that **all words between addresses 0x400 and 0x500 are clearly shown in the window**



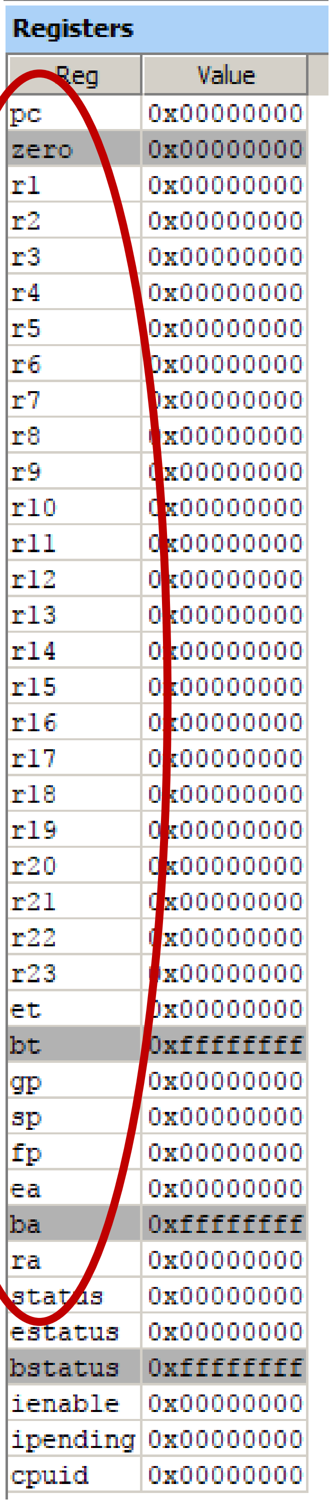
Memory

Goto memory address Address (hex): 400 Go Query All Devices

```
             +0x0      +0x4      +0x8      +0xc
0x00000400   00000001  00000003  00000005  00000007
0x00000410   00000009  0000000b  0000000d  00000007
0x00000420   00000010  00000011  00000012  00000003
0x00000430   b8999999  d19b9198  999f99d9  99999999
0x00000440   99d99fd5  81999199  99991991  1989999b
0x00000450   98919999  99b99159  9999999b  9999bd99
0x00000460   99999999  99999999  b9d899b9  9d989999
0x00000470   999bb99d  90811199  99d99989  8998999b
0x00000480   89999949  b9a99999  9199199f  99899991
0x00000490   99190989  991909bd  198d999b  d9899d99
0x000004a0   99d99999  999999b1  00000174  000001b4
0x000004b0   00000007  000001d4  00000003  00000031
0x000004c0   00000033  00000003  000001d4  00000098
0x000004d0   00000031  00000003  00000004  00000005
0x000004e0   00000006  00000007  00000008  00000000
0x000004f0   00000003  000001d4  00000007  000001b4
0x00000500   99919991  d9199999  9999a91d  99d99d91
```

Disassembly / Breakpoints / Memory / Watches / Trace

- Screenshot 2 of your registers: Make sure all registers, such as pc, r2, ..., sp, and ra, are all clearly shown in the window.



Registers

Reg	Value
pc	0x00000000
zero	0x00000000
r1	0x00000000
r2	0x00000000
r3	0x00000000
r4	0x00000000
r5	0x00000000
r6	0x00000000
r7	0x00000000
r8	0x00000000
r9	0x00000000
r10	0x00000000
r11	0x00000000
r12	0x00000000
r13	0x00000000
r14	0x00000000
r15	0x00000000
r16	0x00000000
r17	0x00000000
r18	0x00000000
r19	0x00000000
r20	0x00000000
r21	0x00000000
r22	0x00000000
r23	0x00000000
et	0x00000000
bt	0xffffffff
gp	0x00000000
sp	0x00000000
fp	0x00000000
ea	0x00000000
ba	0xffffffff
ra	0x00000000
status	0x00000000
estatus	0x00000000
bstatus	0xffffffff
ienable	0x00000000
ipending	0x00000000
cpuid	0x00000000